X-Force: Force-Executing Binary Programs for Security Applications

Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu
Purdue University
Zhiqiang Lin
UT Dallas
Zhendong Su
UC Davis
Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion
Background & Motivation

- Binary analysis
  - The analysis on compiled binary software
    - No source code & symbolic information
    - More challenging than software analysis using source code
      - Control flow graph, variable type

- Binary analysis has many security applications
  - Exposing malware behavior by constructing CFG/CG
  - Identifying and patching security vulnerabilities
Background & Motivation

Existing approaches

- Static analysis (IDA)
  - Examining the code without executing it
- Dynamic analysis (Valgrind, PIN)
  - Testing and evaluation of an application during runtime
- Symbolic analysis (BitBlaze, S2E)
  - Determine what inputs cause each part of the program to execute

<table>
<thead>
<tr>
<th></th>
<th>Good Coverage</th>
<th>Packing &amp; Obfuscation</th>
<th>Precision</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Dynamic</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Symbolic</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion
Design

What is X-Force?

- Dynamic analysis engine that forces a binary to execute
  - Provide no inputs or any environment setup
  - Explore different paths by simply switching the outcome of predicates
Example

-Hijack the name resolution for a specific domain

```c
1 DNSentry *p;
2 void main () {
3    int x = inputInt ();
4    if (C (x))
5        p = (DNSentry *)malloc (...);
6    if (x & CODE_RED) {
7        genName (x, p);
8        hashTablePut (x, p);
9    }
10   ...
11   hashTablePut (... , o); // o is of type T
12   ...
13   s = hashTableGet (y); // y == x through execution
14   if (s)
15      // redirection for the domain specified by s
16      redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20    inputDirectionary ();
21    *(q->name) = ...Lookup (x, date ())...;
22 }
```
Example – Static Analysis

```c
1 DNSSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSSentry *)malloc (...);
6     if (x & CODE_RED) {
7         genName (x, p);
8         hashTablePut (x, p),
9     }
10    ...
11    hashTablePut (...); 0 // o is of type T
12    ...
13    hashTableGet (y);  // y == x through execution
14    if (s)
15        //redirection for the domain specified by s
16        redirection ();
17 }
18
19 void genName (int x, DNSSentry *q) {
20     inputDirectionary ();
21     *(q->name) = ...Lookup (x, date ());...
22 }
```

Truth: Only object fetched at 13 is from either 8 or 11.
Example – Dynamic Analysis

```c
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSentry *)malloc (...);
6     if (x & CODE_RED) {
7         genName (x, p);
8         hashTablePut (x, p);
9     }
10     ...
11     hashTablePut (... , o); // o is of type T
12     ...
13     s = hashTableGet (y); // y == x through execution
14     if (s)
15         // redirection for the domain specified by s
16         redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20     inputDirectionary ();
21     *(q->name) = ...Lookup (x, date ())...;
22 }
```
Example – Symbolic Analysis

```c
# Example of symbolic analysis

void main () {
    int x = inputInt ();
    if (C (x))
        p = (DNSentry *)malloc (...);
    if (x & CODE_RED) {
        genName (x, p);
        hashTablePut (x, p);
    }
    ...
    hashTablePut (... , o); // o is of type T
    ...
    s = hashTableGet (y); // y == x through execution
    if (s)
        \(//\text{redirection for the domain specified by } s\)
        redirection ();
    }

    void genName (int x, DNSentry *q) {
        inputDirectionary ();
        *(q->name) = ...Lookup (x, date ());...;
    }
```
Example – X-Force

```c
1. DNSentry *p;
2. void main () {
3.     int x = inputInt ();
4.     if (C (x)) {
5.         p = (DNSentry *)malloc (...);
6.         if (x & CODE_RED) {
7.             genName (x, p);
8.             hashTablePut (x, p);
9.         }
10.    ...
11.    hashTablePut (... , o); // o is of type T
12.    ...
13.    s = hashTableGet (y); // y == x through execution
14.    if (s)
15.         // redirection for the domain specified by s
16.         redirection ();
17.    }
18. } 
19. void genName (int x, DNSentry *q) {
20.     inputDirectionary ();
21.     *(q->name) = ...Lookup (x, date ())...;
22. }
```

- Provides random inputs
- Assume all 3 predicates go to false branch
- Leads to a non-interesting path
Example – X-Force

```c
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x)) {
5         p = (DNSentry *)malloc (...);
6         if (x & CODE_RED) {
7             genName (x, p);
8             hashTablePut (x, p);
9         }
10     }
11     ...
12     hashTablePut (... , o); // o is of type T
13     ...
14     s = hashTableGet (y); // y == x through execution
15     if (s) {
16         // redirection for the domain specified by s
17         redirection ();
18     }
19     void genName (int x, DNSentry *q) {
20         inputDirectionary ();
21         *(q->name) = ...Lookup (x, date ())...;
22 }
```
Example – X-Force

```c
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSentry *)malloc (...);
6     if (x & CODE_RED) {
7         genName (x, p);
8         hashTablePut (x, p);
9     }
10    ...
11    hashTablePut (... , o); // o is of type T
12    ...
13    s = hashTableGet (y); // y == x through execution
14    if (s)
15        // redirection for the domain specified by s
16        redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20     inputDirectionary ();
21     *(q->name) = ...Lookup (x, date ())...;
22 }
```

Not covered, p = NULL
Flip predicate at line 6
Memory write exception, crash!
Crash-free Execution

- Ideas on memory access exception
  - Skip it?
    - A lot of following exceptions, cascading effect on program state corruption
    - Lose heap data
  - Allocate a piece of memory on demand
    - It is not sufficient by just fixing the corrupted pointer itself
    - Fix the other correlated pointers
Example – Dataflow

```c
1 DNSentry *p;
2 void main () {
3    int x = inputInt ();
4    if (C (x))
5        p = (DNSentry *)malloc (...);
6    if (x & CODE_RED) {
7        genName (x, p);
8        hashTablePut (x, p);
9    }
10    ...
11    hashTablePut (... , o); // o is of type T
12    ...
13    s = hashTableGet (y); // y == x through execution
14    if (s)
15        // redirection for the domain specified by s
16        redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20    inputDictionary ();
21    *(q->name) = ...Lookup (x, date ())...;
22 }
```

Not covered, p = NULL
Flip predicate at line 6
Memory write exception, crash!
Crash-free Execution

- **Observations**
  - Some pointers are correlated
  - Correlated pointers are only linearly correlated
    - No multiplication/division

- **Solution – Linear set tracing**
  1. Memories/registers that are linearly correlated are put into a set
     - Copying (e.g. b = a)
     - Adding or subtracting (e.g. q = p +/- 4)
  2. When memory exception occurs, recover values for elements based on maintained linear sets
Example – Linear Set Tracing

```c
1  DNSentry *p;
2  void main () {
3      int x = inputInt ();
4      if (C (x))
5          p = (DNSentry *)malloc (...);
6      if (x & CODE_RED) {
7          genName (x, p);
8          hashTablePut (x, p);
9      }
10     ...
11     hashTablePut (... , o); // o is of type T
12     ...
13     s = hashTableGet (y); // y == x through execution
14     if (s)
15         //redirection for the domain specified by s
16         redirection ();
17 }
18  void genName (int x, DNSentry *q) {
19      inputDirectionary ();
20      *(q->name) = ...Lookup (x, date ()) ...;
21  }
22 }
```

Memory write exception, crash!

p  q
q->name
Path Exploration

- Exploration algorithms
  - Branch coverage driven algorithm
    - Number of executions - $O(n)$
      - $n$ denotes the number of basic blocks
  - Exponential search algorithm - $O(2^n)$

- Implement a taint analysis subsystem
  - Determine branches that are input related
The Essence of X-Force

- Reachable program state
  - Ideal coverage
- Static analysis
  - Over-approximate coverage
- Dynamic analysis
  - Under-approximate coverage
- X-Force
  - Practicality
The Essence of X-Force

- X-Force is important in practice
  - Results are not affected much by infeasible paths
    - Only a small number of predicates are switched
  - Fast
  - Naturally handle packed, obfuscated, and even self-modifying binaries
  - Existing dynamic analysis can be easily ported to X-Force
Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion
Implementation Challenges

- Indirect jump - Jump Table
  - Leverage existing jump table reverse engineering techniques
  - Treat them as direct conditional branch in exploration algorithms

- Loops
  - If the loop bound is computed from input, it may be a corrupted value
    - Use taint analysis subsystem to determine if it’s input related
    - If so, set the loop bound to a pre-defined constant

- Recursions
  - Maintain call stack during execution to detect recursion
  - If recursion is too deep, skip calling into it by simulating a return instruction
Implementation Challenges

- Handling library function calls
  - I/O functions, memory manipulation functions
- Protect stack memory
  - Return addresses, base pointers
- Handling multiple thread execution
  - Serialize the execution
  - Explore different thread scheduling
Outline

- Background & Motivation
- Design
- Technical Challenges
- Evaluation
- Conclusion
## Evaluation: Case Study I – CFG/CG Construction

### Instruction Coverage

<table>
<thead>
<tr>
<th></th>
<th>IDA</th>
<th>Dynamic</th>
<th>X-Force</th>
<th>Dynamic \ X-Force</th>
<th>X-Force \ Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>7913</td>
<td>3601</td>
<td>5075</td>
<td>0</td>
<td>1474</td>
</tr>
<tr>
<td>175.vpr</td>
<td>31847</td>
<td>19409</td>
<td>29218</td>
<td>0</td>
<td>9820</td>
</tr>
<tr>
<td>176.gcc</td>
<td>310277</td>
<td>157451</td>
<td>227546</td>
<td>0</td>
<td>70095</td>
</tr>
<tr>
<td>181.mcf</td>
<td>2184</td>
<td>1622</td>
<td>1935</td>
<td>0</td>
<td>313</td>
</tr>
<tr>
<td>186.crafty</td>
<td>43327</td>
<td>27811</td>
<td>42763</td>
<td>0</td>
<td>14952</td>
</tr>
<tr>
<td>197.parser</td>
<td>25532</td>
<td>17339</td>
<td>23135</td>
<td>0</td>
<td>5796</td>
</tr>
<tr>
<td>252.eon</td>
<td>70592</td>
<td>15580</td>
<td>27224</td>
<td>0</td>
<td>11644</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>132264</td>
<td>55964</td>
<td>33643</td>
<td>28961</td>
<td>6640</td>
</tr>
<tr>
<td>254.gap</td>
<td>113410</td>
<td>37564</td>
<td>110066</td>
<td>0</td>
<td>72502</td>
</tr>
<tr>
<td>255.vortex</td>
<td>132053</td>
<td>53798</td>
<td>101207</td>
<td>0</td>
<td>47409</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>5761</td>
<td>3612</td>
<td>4830</td>
<td>0</td>
<td>1218</td>
</tr>
<tr>
<td>300.twolf</td>
<td>46556</td>
<td>19996</td>
<td>41935</td>
<td>0</td>
<td>21939</td>
</tr>
</tbody>
</table>
### Evaluation: Case Study I – CFG/CG Construction

#### Indirect Call Edge Coverage

<table>
<thead>
<tr>
<th></th>
<th>IDA</th>
<th>Dynamic</th>
<th>LLVM</th>
<th>X-Force</th>
<th>Dynamic \ X-Force</th>
<th>X-Force \ Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>176.gcc</td>
<td>25</td>
<td>169</td>
<td>9141</td>
<td>1720</td>
<td>0</td>
<td>1551</td>
</tr>
<tr>
<td>252.eon</td>
<td>0</td>
<td>60</td>
<td>28802</td>
<td>121</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>24</td>
<td>225</td>
<td>-</td>
<td>151</td>
<td>122</td>
<td>48</td>
</tr>
<tr>
<td>254.gap</td>
<td>2</td>
<td>1103</td>
<td>187155</td>
<td>20485</td>
<td>0</td>
<td>19382</td>
</tr>
<tr>
<td>255.vortex</td>
<td>0</td>
<td>28</td>
<td>340</td>
<td>30</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
## Evaluation: Case Study I – CFG/CG Construction

### Performance

<table>
<thead>
<tr>
<th></th>
<th>Running Time (s)</th>
<th># of Runs</th>
<th>Avg. Switched Predicates # / Total #</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>704</td>
<td>246</td>
<td>2.1/1291</td>
</tr>
<tr>
<td>175.vpr</td>
<td>8725</td>
<td>1849</td>
<td>4.7/2164</td>
</tr>
<tr>
<td>176.gcc</td>
<td>173241</td>
<td>26606</td>
<td>12.9/29847</td>
</tr>
<tr>
<td>181.mcf</td>
<td>129</td>
<td>113</td>
<td>4.3/153</td>
</tr>
<tr>
<td>186.crafty</td>
<td>43995</td>
<td>2496</td>
<td>8.0/62582</td>
</tr>
<tr>
<td>197.parser</td>
<td>3424</td>
<td>1820</td>
<td>6.4/944</td>
</tr>
<tr>
<td>252.eon</td>
<td>6379</td>
<td>2091</td>
<td>4.1/3146</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>7137</td>
<td>843</td>
<td>8.3/9535</td>
</tr>
<tr>
<td>254.gap</td>
<td>50745</td>
<td>7319</td>
<td>6.0/173316</td>
</tr>
<tr>
<td>255.vortex</td>
<td>34776</td>
<td>8566</td>
<td>7.3/2548</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>557</td>
<td>209</td>
<td>1.4/7001</td>
</tr>
<tr>
<td>300.twolf</td>
<td>10043</td>
<td>2825</td>
<td>5.4/1322</td>
</tr>
</tbody>
</table>
## Evaluation: Case Study II – Malware Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>MD5</th>
<th>File Size(KB)</th>
<th>Number of Library Call Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>dg03.exe</td>
<td>4ec0027bef4d7e1786a04d021fa8a67f</td>
<td>192</td>
<td>808 546 1750</td>
</tr>
<tr>
<td>Win32/PWSteal.F</td>
<td>04eb2e58a145462334f849791bc75d18</td>
<td>20</td>
<td>9 28 94</td>
</tr>
<tr>
<td>APT1.DAIRY</td>
<td>995442f772cc037885335340fc297ea0</td>
<td>19</td>
<td>213 68 236</td>
</tr>
<tr>
<td>APT1.GREENCAT</td>
<td>0c5e9f564115bfcbee66377a829de55f</td>
<td>14.5</td>
<td>303 114 302</td>
</tr>
<tr>
<td>APT1.HELAUTO</td>
<td>47e7f92419eb4b98ff42124c3ca11b738</td>
<td>8.5</td>
<td>109 33 109</td>
</tr>
<tr>
<td>APT1.STARSYPOUND</td>
<td>1f2eb7b090018d975e6d9b40868cs94ca</td>
<td>7</td>
<td>80 15 80</td>
</tr>
<tr>
<td>APT1.WARP</td>
<td>36cd49ad631e99125a3bb2786e405cea</td>
<td>45.5</td>
<td>495 156 414</td>
</tr>
<tr>
<td>APT1.NEWSREEL</td>
<td>2e49f47c98203b110799ab622265f4ef</td>
<td>21</td>
<td>189 49 192</td>
</tr>
<tr>
<td>APT1.GOOGLES</td>
<td>57f98d16ace439a11012860f88db21831</td>
<td>10.5</td>
<td>127 45 131</td>
</tr>
<tr>
<td>APT1.BOUNCER</td>
<td>6ebd05a02459d3b22a9d4a79b8626bf1</td>
<td>56</td>
<td>24 39 562</td>
</tr>
</tbody>
</table>

- X-Force discovers more lib calls than IDA for packed/obfuscated malware
- X-Force beats dynamic native run for all the programs
Evaluation: Case Study III
– Type Reverse Engineering

- REWARDS
  - A dynamic analysis tool of type reverse engineering
- Porting REWARDS to X-Force
  - X-Force provides concrete execution states that are used by REWARDS
  - Little modification
- Results
  - Increase variable coverage from 57% to 84%
  - Increase type reverse accuracy from 88% to 90%
References

- Static analysis
  - Codesurfer/x86
  - IDA-Pro
  - Tie

- Dynamic analysis
  - Dart
  - REWARDS
  - Howard
  - Panorama

- Symbolic analysis
  - KLEE
  - S2E
  - BitBlaze
Outline

- Background & Motivation
- Design
- Technical Challenges
- Evaluation
- Conclusion
Conclusion

- Propose dynamic analysis engine X-Force, a system that can force binary to be executed
  - Requiring no inputs or any environment setup
- Develop a crash-free execution model
  - Detect and recover exceptions properly.
- Develop various execution path exploration algorithms
  - Provide customized options for users to reduce search spaces
- Evaluate X-Force on 3 types of case studies
  - CFG/CG construction
  - Malware analysis
  - Type reverse engineering
Thank you!

Q & A